Universidad de Guayaquil Facultad de Ciencias, Matemáticas y Físicas

Carrera de Ingeniería en Software

Asignatura:

Desarrollo de Aplicaciones Web Avanzado.

Integrantes:

Naranjo Espinoza Jeshua Daniel.

Aguilar Villafuerte Daniel Mateo.

Mendoza Medina Angelica Samanta.

Gonzalez Astudillo Adrian Enrique.

Paralelo:

7 – 1.

Año:

2025 – 2026.

Manual de Usuario: Sistema de Gestión "Ceragen"

1. Introducción

El propósito de este sistema es automatizar y gestionar el proceso de terapias físicas y alternativas para los pacientes de la empresa, solucionando la problemática del seguimiento manual de sesiones, pagos y agendamientos.

El sistema está diseñado bajo una arquitectura de microservicios para asegurar su escalabilidad y adaptabilidad, comprendiendo tres módulos principales:

- **Seguridad**: Gestiona la autenticación, los usuarios, los roles y los permisos del sistema.
- **Administración**: Controla la gestión de personal médico, productos, servicios, gastos y la generación de reportes.
- **Pacientes**: Administra la información de los pacientes, su historial clínico, el registro de sesiones, los pagos y el agendamiento de citas.

2. Arquitectura del Sistema

El backend de Ceragen sigue un patrón de diseño modular que separa las responsabilidades en distintas capas, facilitando la organización, el desarrollo y el mantenimiento.

2.1. Estructura de Directorios del Backend

La estructura del proyecto del backend es la siguiente:

- **ws_ceragen/**
    - **src/**
        - **api/**
            - Components/: Contiene la lógica de negocio y la interacción con la base de datos.

        - Model/: Define los esquemas (modelos) para la validación de las solicitudes.
        - Routes/: Define las rutas (endpoints) de la API.
        - Services/: Actúa como controlador, manejando las solicitudes HTTP y coordinando las respuestas.
    - **utils/**
        - database/: Gestiona la conexión a la base de datos.
        - general/: Almacena la configuración (config.py, config.cfg), y la gestión de logs (logs.py) y respuestas (response.py).
        - pdf/: Contiene la lógica para la generación de documentos PDF (generate_pdf.py).
        - smpt/: Gestiona el envío de correos electrónicos.
  - app.py: Punto de entrada principal de la aplicación Flask.
  - requirements.txt: Lista de dependencias del proyecto.

3. Tecnologías Utilizadas

- **Lenguaje**: Python.
- **Framework Web**: **Flask** y **Flask-RESTful** para la construcción de la API REST.
- **Base de Datos**: **PostgreSQL**, gestionada a través del adaptador **psycopg2**.
- **Autenticación**: **JWT (JSON Web Tokens)** para proteger las rutas y gestionar las sesiones de usuario.
- **Comunicación**: **Flask-Cors** para habilitar el Intercambio de Recursos de Origen Cruzado (CORS) y permitir la comunicación con el frontend.
- **Generación de Documentos**: **FPDF** para crear facturas y otros reportes en formato PDF.

- **Variables de Entorno**: **python-dotenv** para la gestión de la configuración del entorno.

4. Configuración del Entorno

4.1. Variables de Entorno

El archivo src/utils/general/config.cfg contiene las configuraciones esenciales para el funcionamiento del backend:

- [AMBIENTE]:
    - env: Define el entorno de ejecución (ej. DESARROLLO o PRODUCCION).
- [DESARROLLO] / [PRODUCCION]:
    - db_user: Usuario de la base de datos.
    - db_pass: Contraseña para el usuario de la base de datos.
    - db_name: Nombre de la base de datos.
    - db_host: Dirección del host de la base de datos.
    - db_port: Puerto de la base de datos (ej. 5432).
    - secret_jwt: Clave secreta para firmar los tokens JWT.
- [SMTP_UG]:
    - mail, app_password, server_smtp, puerto_smtp: Credenciales y configuración para el envío de correos.

5. Gestión de la Conexión a la Base de Datos

El archivo src/utils/database/connection_db.py es responsable de la conexión con PostgreSQL:

- **conn_db()**: Esta función utiliza los parámetros del archivo de configuración para establecer y devolver una nueva conexión a la base de datos.

- **Clase DataBaseHandle**:
    - **getRecords()**: Ejecuta consultas de tipo SELECT y devuelve un conjunto de resultados. Puede devolver todos los registros, uno solo o un número específico.
    - **ExecuteNonQuery()**: Ejecuta operaciones INSERT, UPDATE o DELETE. Para las inserciones, devuelve el ID del último registro creado.

6. Manejo Estandarizado de Respuestas

El archivo src/utils/general/response.py define funciones para generar respuestas HTTP consistentes en toda la API:

- response_success(datos): Para respuestas exitosas (HTTP 200).
- response_inserted(datos): Para creaciones exitosas de recursos (HTTP 201).
- response_not_found(): Cuando un recurso solicitado no se encuentra (HTTP 404).
- response_error(mensaje): Para errores internos del servidor (HTTP 500).
- response_unauthorize(): Para accesos no autorizados por falta de un token o token inválido (HTTP 401).

7. Resumen de Componentes (Lógica de Negocio)

El directorio src/api/Components/ contiene la lógica de negocio, organizada por entidades, que se encarga de procesar los datos y las solicitudes de la aplicación. A continuación, se detallan las responsabilidades de cada componente principal:

- **UserComponent**: Este componente es central para la seguridad y el acceso al sistema. Sus responsabilidades incluyen:
    - **Login**: Verifica las credenciales (email y contraseña) de un usuario para permitirle el acceso.

    - **Gestión de Usuarios**: Permite crear, consultar, actualizar y desactivar cuentas de usuario.
    - **Recuperación de Contraseña**: Gestiona el proceso de restablecimiento de contraseña, incluyendo la generación de un token temporal y la actualización de la nueva contraseña en la base de datos.
    - **PersonComponent**: Este componente puede manejarse de manera individual se decidió complementarlo a través de UserComponent, se encarga de la gestión completa de las personas en el sistema. Esto incluye la creación, consulta, actualización y eliminación lógica (cambio de estado) de los registros de personas.

- **RolComponent**: Administra los roles del sistema (por ejemplo, Administrador, Secretario, Médico), permitiendo crear nuevos roles, consultarlos, actualizarlos y cambiar su estado.
- **UserRolComponent**: Gestiona la asignación de roles a los usuarios. Permite asociar uno o más roles a un usuario específico, así como actualizar o eliminar dichas asignaciones.
- **InvoiceComponent**: Se encarga de toda la lógica relacionada con la facturación:
    - **Creación**: Genera nuevas facturas con su respectivo número secuencial, calcula el subtotal, descuento, impuestos y total.
    - **Consulta**: Permite obtener una lista de todas las facturas o consultar una factura específica por su ID, incluyendo los detalles del cliente y los productos.
    - **Anulación**: Cambia el estado de una factura a "anulada", registrando quién y cuándo realizó la operación.
    - **Reportes**: Proporciona datos para el dashboard, como las ventas totales del día y un desglose de las ventas por día para la semana actual.
- **PaymentComponent**: Administra los pagos y abonos asociados a las facturas. Permite registrar nuevos pagos (con el método de pago utilizado), consultar los pagos de una factura, actualizarlos y eliminarlos lógicamente.

- **SchedulingComponent**: Gestiona todo lo relacionado con el agendamiento y seguimiento de las sesiones de terapia:
    - **Consulta de Sesiones**: Verifica la cantidad de sesiones disponibles para un paciente según los paquetes que ha comprado.
    - **Agendamiento**: Permite agendar una sesión para un paciente, validando que el horario esté disponible para el terapeuta seleccionado.
    - **Gestión de Citas**: Permite consultar las citas agendadas, actualizarlas o marcarlas como "consumidas" una vez que el paciente ha asistido a la terapia.
- **PatientComponent**: Se enfoca en la gestión de la información clínica y personal de los pacientes. Esto incluye:
    - **Datos del Paciente**: Administra las condiciones médicas, alergias, tipo de sangre y contactos de emergencia.
    - **Historial Clínico**: Permite registrar y consultar el historial de dolencias, tratamientos y notas médicas de cada paciente (MedicalHistoryComponent).

8. Rutas de la API

El archivo src/api/Routes/api_routes.py define todos los endpoints de la API:

- /login: Autenticación de usuarios.
- /Person/list, /Person/create: Gestión de personas.
- /Rol/list, /Rol/create: Gestión de roles.
- /User/list, /User/create: Gestión de usuarios.
- /recover-password, /change-password: Recuperación de contraseña.
- /invoice/create: Creación de facturas.
- /invoices: Listado de todas las facturas.
- /payment/create: Registro de pagos.
- /dashboard/today-sales: Reporte de ventas del día.
- /patient-sessions: Consulta de sesiones disponibles para un paciente.

- /schedule-session: Agendamiento de una sesión.
- /consume-session: Marcar una sesión como utilizada.

9. Punto de Entrada Principal (app.py)

El archivo app.py inicializa y configura el servidor Flask:

- **Inicialización**: Crea la instancia de la aplicación Flask.
- **Configuración de CORS**: Permite las solicitudes desde los orígenes definidos.
- **Carga de Rutas**: Monta todos los endpoints definidos en api_routes.py.
- **Swagger UI**: Configura una interfaz de usuario interactiva para la documentación de la API, accesible en la ruta /ws/secoed/.
- **Inicio del Servidor**: Inicia el servidor para que escuche las solicitudes entrantes en el puerto configurado.